



Finalização de Jogo do Gênero Puzzle



cho que você percebeu que para a criação de um jogo, completo e finalizado precisaríamos de muito mais tempo para a sua execução. Porém, aprendemos bastante coisa, não é?!

Apesar dessa aula ter o nome finalização, não seremos capazes de finalizar completamente esse jogo na aula. Seria importante pensar em interfaces, pontuação e outros elementos para ele ficar comercial. Porém, ele foi um ótimo exercício para aprendermos as etapas de desenvolvimento do jogo bem como a sua mecânica.

Nessa aula iremos aprender:

- **Criação de Game Manager:** Aprender a fazer o jogo buscar novo personagem para atacar inimigos;
- **Desenvolvimento de Game Manager:** Vincular personagens e mecânica ao gerenciador do jogo;
- **Correção Drag e Câmera:** Corrigir elementos de código para que o gerenciador funcione adequadamente;
- **Revisão:** Rememorar pontos importantes aprendidos durante a disciplina.

- **Criação de Game Manager**



Continuando o jogo, precisaremos criar um Game Mananger para que os personagens possam ser utilizados mais que uma vez para a destruição total dos objetos.

Para tanto, será necessário que você crie um arquivo de código C# nomeando-o como “GAMEMANAGER”, assim você poderá começar a fazer o código. O código será pensado em 3 etapas:

- Fazer que o jogo reconheça a quantidade de personagens estão em cena;
- Fazer que ele puxe o próximo personagem quando um for atacado;
- Fazer que o personagem, elástico e câmera funcione conectado à GAMEMANEGER.

Vamos começar criando e colocando as primeiras variáveis e métodos na GAMEMANAGER, arquivo de C#:

- Variáveis
- public static GAMEMANAGER instance;
- public GameObject[] espartano;
- public int espartanosNum;
- public int espartanoEmCena = 0;
- private Transform pos;
- public bool win;
- public bool jogoComecou;



Após isso precisaremos criar uma void Awake (Figura 1), esse void é chamado quando a instância do script está sendo carregada, ou seja, o Awake é chamado quando um GameObject ativo que contém o script é inicializado quando uma cena é carregada, ou quando um GameObject anteriormente inativo é definido como ativo, ou depois que um GameObject criado com Object.Instantiate é inicializado (UNITY, 2022).

```
void Awake()
{
    if (instance == null)
    {
        instance = this;
        DontDestroyOnLoad(this.gameObject);
    }
    else
    {
        Destroy(gameObject);
    }

    SceneManager.sceneLoaded += Carrega;
}
```

Figura 1 – void Awake | Autor, 2022

Depois iremos criar um método chamado Carrega que irá puxar informações do "usign UnityEngine.SceneManagement;", lembro que essa linha deve estar no início do código, antes da classe principal (Figura 2). É também por causa desse item que o "SceneManager.sceneLoaded += Carrega" é declarado em Awake.

```csharp
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GAMEMANAGER : MonoBehaviour
{
```

Figura 2 – SceneManagement | Autor, 2022

Adicione um Game Object e tagueie-o como "pos", para criar a tag é só clicar no Inspector>Tag e adicionar. Ele será referenciado no código como contendo a posição do personagem espartano (Figura 3).

```csharp
void Carrega (Scene cena, LoadSceneMode modo)
{
        pos = GameObject.FindWithTag("pos").GetComponent<Transform>();

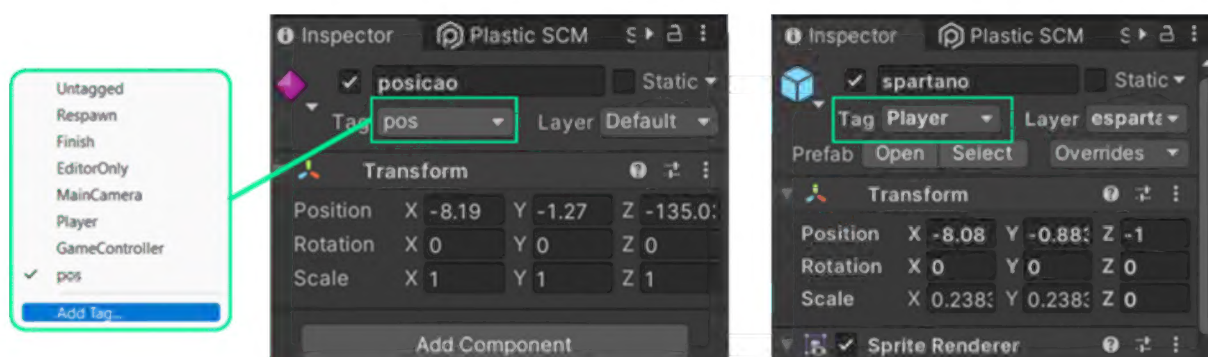
        //Espartano Pos

        espartanosNum = GameObject.FindGameObjectsWithTag("Player").Length;
        espartano = new GameObject[espartanosNum];

        for (int x = 0; x < GameObject.FindGameObjectsWithTag("Player").Length; x++)
        {
            espartano[x] = GameObject.Find("sparta" + x);
        }

}
```

Figura 3 – Método Carrega



Repare que usei como Game Object a Tag “Player”, foi assim que eu criei o tag para nosso personagem (Figura 4).



Perceba ainda que eu criei um laço do tipo “for” para que o código contabilize a quantidade de objetos com a tag “Player”. Além disso, esse código fará que o Game Manager reconheça cada personagem (Figura 5).

No código foi necessário usar o elemento “FindWithTag”, esse elemento retorna um GameObject ativo marcado com tag. Retorna null se nenhum GameObject foi encontrado. Lembre-se que para funcionar é necessário que você deve declarar a tag no gerenciador de tags em Inspector no Unity.



Figura 5 – Reconhecimento do personagem pelo código | Autor, 2022

Agora iremos criar um método que garantirá que os nossos personagens estejam no lugar certo para que sejam lançados contra a muralha e inimigos. Para isso, iremos criar o “void NascSparta” (Figura 6) dentro do código GAMEMANAGER.

Precisaremos colocar também uma variável “string” no começo “public string nomePersonagem;”. Pois ele nos ajudará na construção do método que fará nascer o espartano para jogar contra os obstáculos.

2. **Desenvolvimento de Game Manager**



Esse método fará o seguinte:

Depois poderíamos colocar outros métodos como um Game Over, WinGame e StarGame etc. (Figura 7). Além de colocar o código para rodar em Update.

- Se o espartano for diferente de null;
- Ele irá verificar a posição dele e se for diferente da posição do Pos, ou seja, a posição em que atiramos;
- Ele pegará o nome do personagem;
- Transforma ele para a posição correta;
- E quebra com o próprio código tornando espartanoEmCena como 1.

```
void NascSparta ()
{
    if (espartanoEmCena == 0 && espartanosNum > 0)
    {
        for (int x = 0; x < espartano.Length; x++)
        {
            if (espartano[x].transform.position != pos.position && espartanoEmCena ==0)
                {
                nomePersonagem = espartano[x].name;
                espartano[x].transform.position = pos.position;
                espartanoEmCena = 1;
            }
        }
    }
}
```

Figura 6 – Método NascSparta | Autor, 2022

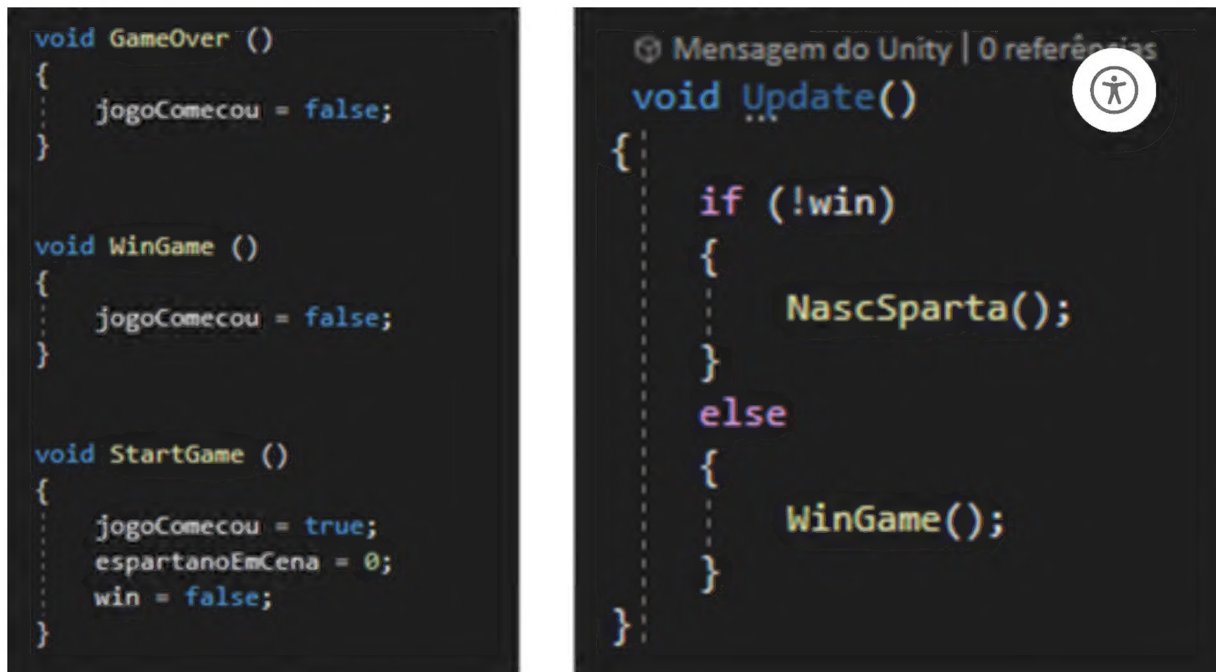


Figura 7 – Métodos e Update | Autor, 2022

3. **Correção de Drag, Câmera e GAMEMANAGER**

3.1 Drag

Agora para a ação funcionar será necessário alterar algumas coisas o código C# Drag. É muito comum que tenhamos que fazer pequenos ajustes para que o código funcione durante todo o jogo.

Abra o código e faça a seguinte alteração:

- Crie a variável “public Rigidbody2D CatapultRB;” e “public bool estouPronto = false;” (Figura 8).

- Depois crie um método Awake e transfira todo o conteúdo que está em “Start” para ela, com exceção da variável “SetupLine();”.

-

No arquivo de código GAMEMANAGER, você precisará trocar de private para public “Transform pos;”.





Agora precisaremos fazer algumas alterações (Figura 8).

```
// variável para usar mais que um espartano
public Rigidbody2D CatapultRB;
public bool estouPronto = false;
```

Entre as alterações será necessário a criação de algumas linhas de códigos, conforme podemos perceber na Tabela 1 além da mudança do componente “spring = GetComponent<SpringJoint2D>();” que foi levado para o início do método (Figura 9).

Código C#
lineFront = (LineRenderer)GameObject.FindWithTag("LF").GetComponent<LineRenderer>();
lineBack = (LineRenderer)GameObject.FindWithTag("LB").GetComponent<LineRenderer>();
CatapultRB = GameObject.FindWithTag("LB").GetComponent<Rigidbody2D>();
spring.connectedBody = CatapultRB;

Tabela 1 – Código para inserir na Void Awake | adaptado de NASCIMENTO, 2022.

Além disso será necessário fazer um ajuste fino com a inclusão de um “Vector2” e a variável “temp”, que estará relacionada com uma spring (Figura 9).

```
Mensagem do Unity | 0 referências
void Awake()
{
    //AJUSTE aula 16:
    spring = GetComponent<SpringJoint2D>();
    lineBack = (LineRenderer)GameObject.FindWithTag("LB").GetComponent<LineRenderer>();
    lineFront = (LineRenderer)GameObject.FindWithTag("LF").GetComponent<LineRenderer>();
    CatapultRB = GameObject.FindWithTag("LB").GetComponent<Rigidbody2D>();
    spring.connectedBody = CatapultRB;

    Vector2 temp = spring.connectedAnchor;
    temp.x = 0;
    temp.y = 0;
    spring.connectedAnchor = temp;

    //importado de void Start
    drag = GetComponent<Collider2D>();
    leftCatapultRay = new Ray(lineFront.transform.position, Vector3.zero);
    espartanoCol = GetComponent<CircleCollider2D>();
    //spring = GetComponent<SpringJoint2D>(); Aula 16 conteúdo foi levado para cima
    spartaRB = GetComponent<Rigidbody2D>();

    catapult = spring.connectedBody.transform;
    rayToMT = new Ray(catapult.position, Vector3.zero);

    audioEsparta = GetComponent<AudioSource>();
}
```

Figura 9– Novidades no código em ajuste da Drag | Autor, 2022

Em void LineUpdate você precisará ajustar criando um condicional "if (transform.name == GAMEMANAGER.instance.nomePersonagem)" colocando todo o conteúdo dentro dela (Figura 10).

```
1 referência
void LineUpdate()
{
    // AJUSTE aula 16: Criar condicional if (transform.name == GAMEMANAGER.instance.nomePersonagem) {} colocar conteúdo dentro dela.

    if (transform.name == GAMEMANAGER.instance.nomePersonagem)
    {
        catapultToSparta = transform.position - lineFront.transform.position;
        leftCatapultRay.direction = catapultToSparta;

        pointL = leftCatapultRay.GetPoint(catapultToSparta.magnitude + espartanoCol.radius -1);

        lineFront.SetPosition(1, pointL);
        lineBack.SetPosition(1, pointL);
    }

}
```

Figura 10 – Ajuste em void LineUpdate | Autor, 2022

Também foi necessário o ajuste do método void SpringEffect, acrescentando na condicional o elemento "&& GAMEMANAGER.instance.espartanoEmCena > 0". Além disso, foi criado uma condição "else if" (figura 11).

```
1 referência
void SpringEffect()
{
    //AJUSTE aula 16: acrescentei  "GAMEMANAGER.instance.espartanoEmCena > 0"
    if (spring != null && GAMEMANAGER.instance.espartanoEmCena > 0)
    {
        if (spartaRB.isKinematic == false)
        {
            if (prevVel.sqrMagnitude > spartaRB.velocity.sqrMagnitude)
            {
                lineFront.enabled = false;
                lineBack.enabled = false;
                Destroy(spring);
                spartaRB.velocity = prevVel;
            }
        }
        //AJUSTE aula 16: acrescentei  esse else if...
        else if (spartaRB.isKinematic == true && transform.position == GAMEMANAGER.instance.pos.position)
        {
            lineFront.enabled = true;
            lineBack.enabled = true;
        }
    }
}
```

Eu acrescentei alguns códigos para viabilizar o uso do mouse no lugar do touch, ou seja, o jogo poderia ser jogado no computador ou no celular. Você poderá ver com calma o código quando baixá-lo, leia os meus comentários, serão importantes para você compreender o que fiz e por quê.

3.2 Câmera

Para fazer a câmera seguir os personagens na tela você precisará refazer basicamente todo o código da "CameraSegue", além de precisar incluir elementos nos arquivos "Drag" e "GAMEMANAGER".

Em Drag você precisará colocar uma condição "if" no método Update (Figura 12).

```
if (spartaRB.isKinematic == false)
{
    Vector3 posCam = Camera.main.transform.position;
    posCam.x = transform.position.x;
    posCam.x = Mathf.Clamp(posCam.x, GAMEMANAGER.instance.objE.position.x, GAMEMANAGER.instance.objD.position.x);
    Camera.main.transform.position = posCam;
}
```

Figura 12 – Condição "if" no método Update

No método Dragging foi necessário criar uma condição para que a Câmera funcionasse junto com o mouse. (Figura 13)

```
1 referência
void Dragging()
{
    // Aula 10: criação de condição

    if(spartaRB.isKinematic)
    {
        Vector3 mouseWP = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        mouseWP.z = 0f;

        catapultToSparta = mouseWP - catapult.position;

        if (catapultToSparta.sqrMagnitude > 9f)
        {
            rayToMT.direction = catapultToSparta;
            mouseWP = rayToMT.GetPoint(3f);
        }

        transform.position = mouseWP;
    }
}
```

Figura 13 – Condição “if” no método Dragging

Nos códigos novos referente ao controle pelo mouse foi acrescentado elementos para o controle de câmera (Figura 14).

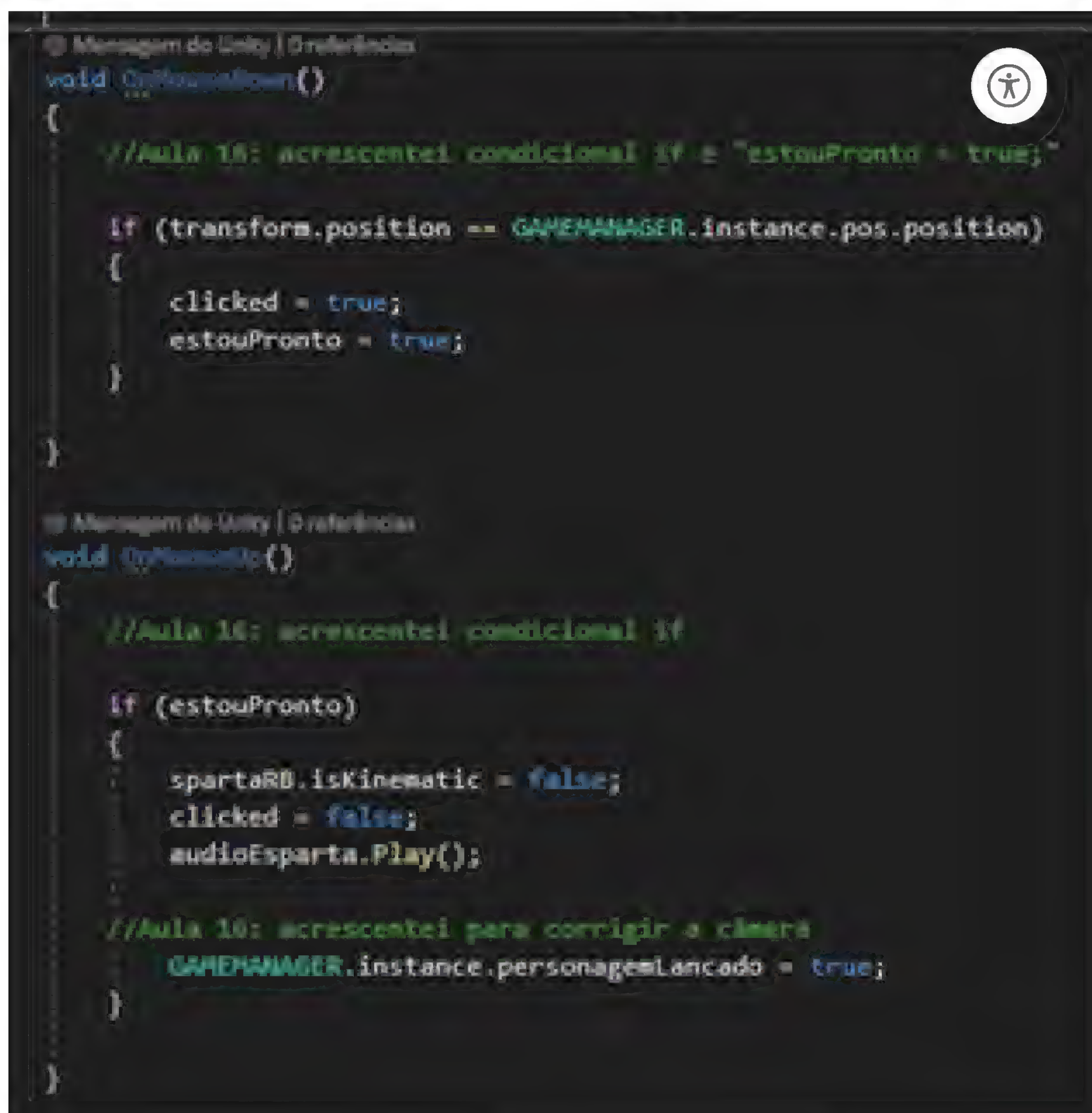


Figura 14 – Métodos OnMouseDown e OnMouseUp

3.3 GAMEMANEGER

Durante a correção da Drag e da Câmera para que o código da GAMEMANAGER funcionasse, precisamos acrescentar alguns outros elementos. No final dessa aula você poderá fazer o Download de todos os códigos com seus comentários (Figura 15).

```
//Variáveis
public static GAMEMANAGER instance;
public GameObject[] espartano;
public int espartanosNum;
public int espartanoEmCena = 0;
//AJUSTE aula 16: troquei de private para public "Transform pos;"
public Transform pos;
public bool win;
public bool jogoComecou;
public string nomePersonagem;

//Aula 16: incremento para controlar a câmera
public bool personagemLancado = false;
public Transform objE, objD;
```

Figura 15 – Alterações para fazer a câmera e o drag funcionarem

```
void Carrega (Scene cena, LoadSceneMode modo)
{
    pos = GameObject.FindWithTag("pos").GetComponent<Transform>();

    //Aula 16: Controle da Câmera
    objE = GameObject.FindWithTag("PE").GetComponent<Transform>();
    objD = GameObject.FindWithTag("PD").GetComponent<Transform>();
    StartGame();

    //Espartano Mov

    espartanosNum = GameObject.FindGameObjectsWithTag("Player").Length;
        espartano = new GameObject[espartanosNum];

        for (int x = 0; x < GameObject.FindGameObjectsWithTag("Player").Length; x++)
        {
            espartano[x] = GameObject.Find("spartano" + x);
        }
}
```

Figura 15 – Alterações para fazer a câmera e o drag funcionarem

Em GAMEMANEGER foi necessário acrescentar mais alguns elementos para o controle de câmera. Na verdade, é aqui que o efeito câmera vai funcionar de forma mais agressiva. Comece acrescentando as variantes:

- public bool personagemLancado = false;
- public Transform objE, objD;

Depois é a hora de Colocar em "void Carrega" dos objetos de orientação de câmera usando as Tags para localizá-las (Figura 16).

```
void Carrega (Scene cena, LoadSceneMode modo)
{
    pos = GameObject.FindWithTag("pos").GetComponent<Transform>();

    objE = GameObject.FindWithTag("PE").GetComponent<Transform>();
    objD = GameObject.FindWithTag("PD").GetComponent<Transform>();
    StartGame();

    espartanosNum = GameObject.FindGameObjectsWithTag("Player").Length;
        espartano = new GameObject[espartanosNum];

        for (int x = 0; x < GameObject.FindGameObjectsWithTag("Player").Length; x++)
        {
            espartano[x] = GameObject.Find("spartano" + x);
        }
}
```

Figura 16 – Controle de Câmera

Já o código "CameraSegue" foi totalmente reescrita, perceba na imagem abaixo Figura 17):

Figura 17 – Código C# CameraSegue | Autor, 2022

Pode ser que eu tenha me esquecido de escrever alguma alteração, portanto, baixe o projeto, abra-o na Unity e perceba cada detalhe do que eito. Sempre que aparecer um erro, leia-o atentamente e vá até a linha em que está o problema, reflita e corrija, até você se habituar com a linguagem, pode parecer difícil, mas com o tempo você irá dominar o Unity e o C#.

4. **Revisão**

É claro que para fazer um jogo comercial, seria necessário fazer outros levels, botões de controle, play, pause, personagens, cenários etc. Não tivemos tempo para isso, mas vocês terão disciplinas que abordarão esse assunto. Nesse momento nosso objetivo era integrar tudo o que aprenderam até aqui. Para tanto, integramos conhecimentos de Game Design, Concept Art e Illustração, UX Design, Engenharia de Requisitos e de Software, C# e Unity.

Entre as coisas que aprendemos podemos citar:

- **Engenharia de requisitos:** aprendemos que a engenharia de requisitos analisa aquilo que será necessário saber e ter de tecnologia para o jogo acontecer;
- **Definição de stakeholders:** entender a importância dos públicos-alvo do jogo para o planejamento e sucesso dele;
- **Customer Centricity:** aprendemos que os negócios do século XXI precisam estar centrados no cliente/consumidor;
- **Game Design Canvas:** um meio de planejar o jogo de forma rápida e objetiva inspirado no Business Model Canvas;
- **Storytelling:** Formas de se conter histórias envolventes, podemos nos basear na jornada do herói;

- **UML:** aprendemos a usar os diagramas como meio de planejamento e formulação de jogos digitais e sistemas;

- **Desenvolvimento de personagens e cenários:** aprender usar softwares de computação gráfica para desenvolvimento de personagens e cenários;

- **Animação:** fazer animação de cenário, câmera e personagens usando o Unity e C#.

- **C#:** Usar o código na programação de jogos digitais;

- **Criar um jogo do gênero puzzle.**

Espero que tenha gostado do conteúdo, muito sucesso para ti, até uma próxima. Vida longa e próspera.

Atividade extra

Assista ao vídeo no Youtube com o tema "Storytelling na construção de jogos digitais". Esse vídeo é do Núcleo de Práticas de Engenharia de Software Aplicadas a Game da UFRRJ. Disponível no Youtube.

Referência Bibliográfica

NASCIMENTO, W. **Jogos 2D com Unity e C#**. São Paulo: Udemy, 2022.

UNITY. **Unity Documatation.** 2022. Disponível em: <https://docs.unity3d.com/>.



Atividade Prática: Finalização de Jogo do Gênero Puzzle

Título da Prática: Finalização de Jogo do Gênero Puzzle

Objetivos: Finalizar a mecânica do jogo para que se possa utilizar mais que um herói para a destruição dos objetos.

Materiais, Métodos e Ferramentas: Para realizar esta prática vamos utilizar o software Unity, um smartphone Android e acesso ao Youtube.

Atividade Prática

Roteiro

1º Passo – Leia o texto do material de apoio.

2º Passo – Assista aos vídeos da disciplina e aula 16.

3º Passo – Usando os passos a passos explicado e feito nos vídeos e Material de Apoio, criar a mecânica de se utilizar mais que um personagem para a destruição da muralha e inimigos.

4º Passo – Baixe os códigos para ler os comentários e ver o que foi feito no link: https://bit.ly/3Hi9S0x.

5º Passo – Adicione ruídos de quebra ao seu objeto.

6º Passo – Repita isso ao seu inimigo.

7º Passo – Verificar se não existe erros e como corrigi-los.

8º Passo – Crie um relatório de tudo o que você fez, bem como erros que surgiram e como você os resolveu.

9º Passo – Caso você ainda sinta dificuldade, baixe a pasta com todo o jogo no link: https://bit.ly/3t17Tsx.

10º Passo – Você poderá acrescentar elementos e UI para deixar o seu jogo mais completo, use seus conhecimentos e criatividade para isso.

Após a conclusão da Atividade salve seu projeto em uma pasta zipada, a poste e responda:

1- Quais outros elementos você acrescentou ao jogo e por quê?

2- Quão difícil você achou o desenvolvimento e por quê?

Gabarito Atividade Prática

1- Você pode ter acrescentado UI, botões e textos de Game Over, Vitória etc. Provavelmente fez isso para deixar o jogo mais interativo e interessante como projeto final.

2- O projeto tem um grau de complexidade grande, principalmente de fizermos outras cenas e levels. Por isso, é necessário que tenhamos a proatividade de buscar mais conhecimentos, aprendermos durante o processo e buscarmos mais conteúdos na internet. Existem vários tutoriais no Youtube, Vimeo etc. Você pode continuar evoluindo!

Ir para exercício